# 制御システムI：センサによる位置決め制御
## ～組み込みプログラミングによるロボット制御～

## 1　はじめに

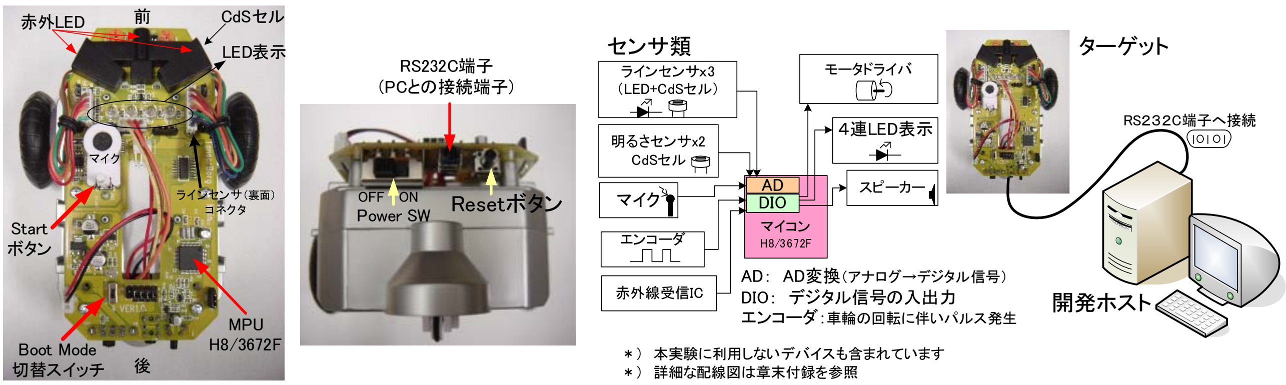
マイクロプロセッサ（マイコン）などによるコンピュータ制御によって動作している機器を**組み込み機器／装置**，**組み込みシステム**などと呼ぶ．昨今では，身の回りの多くの電気機器がデジタル化されつつあり，たとえば，テレビ，ビデオデッキなどのマルチメディア関連機器，冷蔵庫，洗濯機，エアコン，電子レンジなどのマイコン制御による家電製品，インターネットへの接続を担うルータ機器，ネットワークスイッチなどの通信機器，携帯電話などの情報端末，ロボット，自動車 等々，挙げればきりがないほどの電子電気機器が組み込みコンピュータを内蔵した装置に位置づけられる．それらの装置に内蔵されているマイクロプロセッサ上で動作するソフトウェアのことを**組み込みプログラム**と呼ぶ（あるいは**ファームウェア**とも良く言われる）．

最近のマイクロプロセッサの小型化，多機能化の進展は目覚ましいものがあり，ワンチップのプロセッサ（ワンチップマイコン）でセンサ類やモータ類などを制御できる機能を内蔵したもの，さらにはプログラムを格納しておくための ROM(最近では書き換え可能なフラッシュメモリ) やプログラム動作に必須の RAM までも内蔵しているものが普及している．

本実験では，マイクロプロセッサを内蔵した簡単な移動型ロボットを対象として，それに組み込むプログラムの開発工程（クロス開発という）を理解するとともに，実際にセンサ信号の入力，モータ駆動を伴う位置決め制御プログラムの作成を通して，組み込み機器の動作原理とその制御システムの構成の全体像を理解することを目的とする．実験第 1 日目は，搭載された光学センサを利用して，ロボットに床面に描いた黒線をたどって走行させるライントレース動作の実験を通して，プログラム開発の流れを理解する．実験 2 日目は，ロボットに搭載された光学センサの信号の評価，及びロボットの位置決め制御実験を行う．

## 2　実験に利用する移動ロボットのハードウェア構成

ロボット (図 1) は教育玩具用に市販されているものであるが，組み込み装置向けに広く普及しているルネサステクノロジ社製の H8-Tiny シリーズのマイクロプロセッサ（H8/3672 型番：HD64F3672FP）を搭載している．本プロセッサは，16 ビット高速 CPU であり (動作周波数 16MHz)，ROM（フラッシュメモリ）を 16KB, RAM を 2048 バイト（2KB）を内蔵しているため外付けのメモリを必要とせず，ワンチップ動作が可能である．また AD 変換機能を内蔵しているため，アナログ出力のセンサ類を直接接続することができる．またデジタル信号の入出力が可能であり，スイッチ類や LED などの表示装置と簡単に接続できる．すなわち，図 1(c) に示すように今回のロボットのほぼ全機能がワンチップで管理可能となっている．(詳細な回路図は章末付録 A を参照)



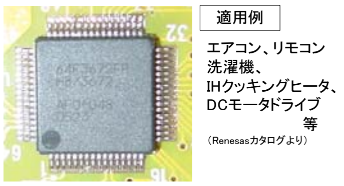
(a) Top View　(b) 後部　(c) 簡単な内部接続図　(d) PCとの接続

図 1: 実験に利用する移動型ロボットと接続構成

# 3　プロセッサの概略とソフトウェアのクロス開発環境

## 3.1　H8 プロセッサの概略

ルネサステクノロジ社製の MPU（マイクロプロセッサユニット）の SuperH シリーズや H8 シリーズは組み込み装置向けに広く普及し，家電製品，ネットワーク装置，カーナビ，ロボット，プリンタ装置など多くの身近な製品群の動作を担っている．実験に利用するロボットに搭載している H8-Tiny シリーズの MPU は，その中でも小規模システム向けに製品化されたものであり，モータ駆動装置やエアコン，電子レンジや冷蔵庫などの家電製品，セキュリティ機器などへの応用を想定して開発されている．




(a) H8/3672F

・16ビット高速H8/300H CPU
・ROM: 16KB，RAM 2KB 内蔵
・汎用レジスタ　16ビット×16本
・基本命令　62種類
・各種タイマ機能内蔵
・10ビットAD変換器内蔵（入力：0〜5V）
・汎用入出力ポート　26本
・シリアル通信機能内蔵

(b) 主な仕様

図 2: MPU の外観と主な仕様

本実験装置に搭載している H8/3672F（型番：HD64F3672FP）は，ROM（フラッシュメモリ）を 16KB, RAM を 2048 バイト（2KB）を内蔵している．パソコン上で動作するプログラムについては，OS がメモリ管理を行ってくれるため，ユーザがメモリ領域の使い方を意識することは少ないが，組み込み向けプロセッサの場合には ROM と RAM の使い分けを意識することが望ましい．ROM(Read Only Memory) のデータは，通常では変更することができないが，電源をオフにしてもデータが維持される．RAM(Random Access Memory) に格納するデータは自由に変更可能であるが，電源オフでデータは消滅する．（MPU 上の ROM は，大抵の場合フラッシュメモリであり，一定の手続きを踏んでプログラムを書き換えることができるようになっている）

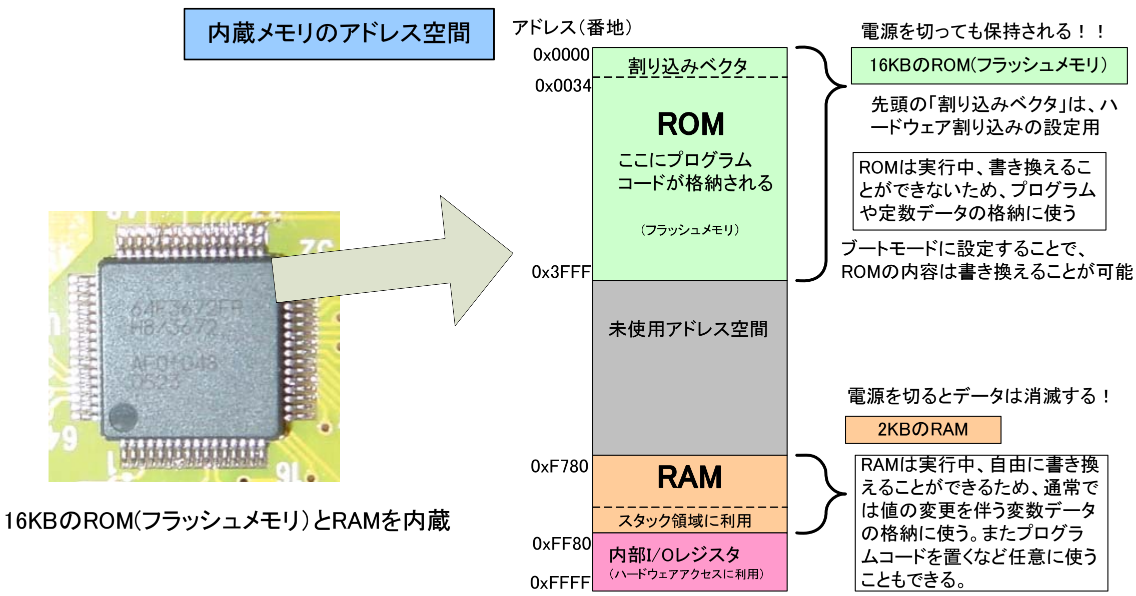



図 3: HD64F3672FP のメモリアドレス空間とその利用

一般的な ROM と RAM の使い方として (図 3 参照)，ROM の前半にプログラムコード（命令コード）のセクション（領域，P セクションと呼ぶ），次に値の変化しないデータ（定数変数）を格納するセクション（C セクション），そして初期値を持つ変数用の初期値データを格納しておくセクション（D セクション），ここまでは

ROM に格納する．初期化を必要としない変数用のセクション（B セクション）は RAM 領域に置かれ，初期化を必要とする変数は実行段階において D セクションを RAM 領域にコピーして利用する．また，関数の中で使うローカル変数は，RAM 領域のスタック領域と呼ばれるセクションを用意して，必要に応じて（動的に）生成し，必要なくなると解放する．MPU によってメモリサイズや ROM ／ RAM サイズが異なるため，これらのメモリ領域の割り当てはコンパイラ環境などで MPU に合わせて任意に指定することができる．(本実験では，事前に設定済である）

## 3.2　クロス開発環境

組み込み装置の場合，PC のようにディスプレイやキーボードなどを備えているわけではない．また使い勝手の良い OS を実行するには非力である (開発プログラムの移植性を高めるため，組み込みシステム向けの OS もある：Windows Embedded, Linux, VxWorks, ITRON etc.)．大抵の場合，当該装置上でプログラムを開発することはしないで，組み込み装置上で動作する機械語コードまでを PC などのプログラム開発用の別のコンピュータ上で開発する．このような開発形態は**クロス開発**とも言われる．その流れを図 4 に示している．

本実験で利用する開発環境は，ルネサステクノロジ社から提供されている統合開発環境 HEW4(High-performance Embedded Workshop Version.4) を利用する.（有償ソフトウェアであるが，64KB 未満の実行コードであれば試用期間を越えて利用できる）．その他，フリーソフトウェアの gcc でもクロス開発が可能であり，Windows だけでなく Linux 環境等の UNIX OS でも無償で開発が行える．

作成された機械語コード（実行コード）は，RS232C 接続や USB，あるいはネットワークなどの通信を介するか，特別な書き込み装置などを経て，組み込み装置上の ROM（書き換え可能なフラッシュメモリ）に書き込むことで，当該機器が動作するようになる．本実験で利用する MPU は，RS232C 接続（シリアル通信）により PC と直接接続できる機能を内蔵しているため，作成したプログラムは PC 上から書き込むことができる．書き込みツールは，同じくルネサステクノロジ社から提供されている FDT(Flash Development Toolkit Version.3.1, 今回利用するのは unsupported version) と呼ばれるソフトウェアを利用する．

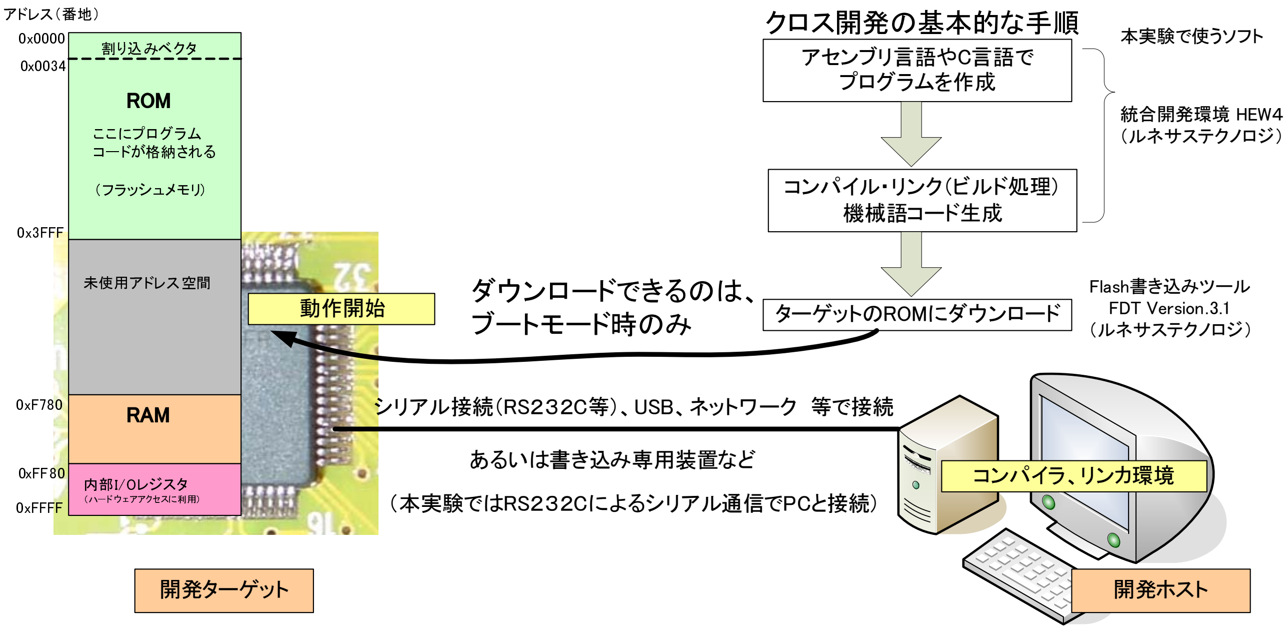



図 4: クロス開発環境

# 4　プログラムを動作させるまでの一連の流れ (ROM 版)

動作プログラムをすべて MPU の ROM 上に格納して動作させる方法を述べる．実行プログラムの最終版が完成した後，以下の手順を踏むことで，電源を投入後いつでもそのプログラムの動作をさせることができる．

### 【1】C 言語ソースコードなどのテンプレートファイルをコピー

各 PC のデスクトップに「ロボット制御実験用テンプレート」フォルダが置いてあるので，同じくデスクトップの「光システム実験２」の中の午前グループは「AM」，午後グループは「PM」フォルダを開き，そこにコピーする．そして，フォルダの名前を「ロボット制御実験用テンプレート」から，実験した日付「X 月 Y 日」に変更する．

### 【2】統合開発環境 HEW4 の起動

コピーしてきたフォルダを開いて,「RobotJikken.hws」をダブルクリックして起動する．そのファイルには開発するターゲット MPU の情報やコンパイル環境，及びメモリ空間の利用設定など必要な設定が保存してあり，その設定が読み込まれた上で HEW4（図 5）が起動する．(フォルダが移動されましたとの主旨のウィンドウが出たら，そのまま OK とする）

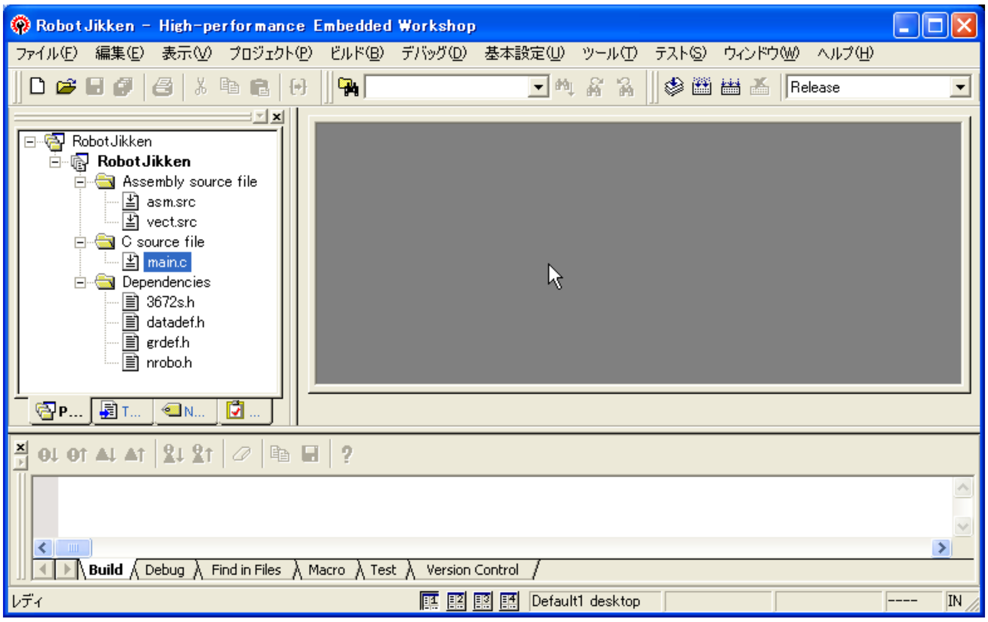



図 5: HEW4 の起動画面

左上側の枠にテンプレートのプロジェクトで利用しているファイルがツリー形式で表示されている．プログラムの起動を担う部分のみアセンブリ言語で記述してあり”Assembly source file” に入っている．メインとなるプログラムは”C source file” に分類されている「main.c」である．main.c をダブルクリックすると，右の枠内に C 言語ソースが表示される．

### 【3】main.c の処理の流れの理解とロボット動作プログラムの追記

ロボットの電源投入，あるいはリセットがかかると，最低限の初期化処理の後，C 言語で書かれた main 関数に処理が移される（ここまでの処理はアセンブリ言語で記述されている．ソースファイル asm.src 参照）.

main 関数の冒頭の初期設定の部分は削除しないこと．各種レジスタの初期化，グローバル変数の初期化などを行っている．`while(1)` の部分から無限ループがはじまり，その後の `while(1)` … の部分でロボット上の Start ボタンが押されるまで無限に待つための記述 1 行がある．しがたって,「ここから‥記述」と書いた部分のプログラムは Start ボタンが押されると実行がはじまる．

以下の**「ここから‥記述」〜「ここまで」の部分を追記する**．この処理は，単に４つの LED を全灯／消灯を無限に繰り返す処理である．Led_Pat() 関数はあらかじめ用意している関数であり，引数に０〜0xF（16 進数）を指定すると，ビットが１に対応する LED が点灯する．

**main.c の基本構成： サンプル１**

```
///////////////////////////////////////////////////
// 千歳科学技術大学　光システム実験
//  テンプレートソースコード
///////////////////////////////////////////////////
#include "3672s.h"//マイコンの機能定義ファイル
#include "nrobo.h"//定数の定義ファイル
#include "grdef.h"//関数・変数の定義ファイル
#include "datadef.h"//データテーブルの定義ファイル
#include <machine.h>//Renesas 提供組込み関数の利用

int main(void)
{
        //////////////////////////////////
        // 初期設定（削除しないこと）
        initVariables();
        initMPU();
        Act_Stop();//動作全停止
        initLED();;//4 連 LED 順次点灯表示
        //////////////////////////////////

        while(1){
                initLED();
                //Start ボタンが押されるまで待ち
                while(1){ if(start_sw_det()!=OFF) break;}

                ///////////////////////////////////////////
                flag_led_on=ON;//LED 関係のセンサ開始
                flag_user=ON;//動作開始フラグをオン
                Wait(30);//センサ出力安定まで少々待ち

                ///////////////////////////////////////////ここから
                // ここからロボットの動作プログラムを記述
                ///////////////////////////////////////////
                while(1){
                        Led_Pat( 0xF );// 4 連 LED をすべて ON
                        Wait( 100 ); //100 x 10ms = 1 sec の待ち
                        Led_Pat( 0x0 );// 4 連 LED をすべて OFF
                        Wait( 100 );
                }
                ///////////////////////////////////////////ここまで
        }
}
  ・
  ・
  ・
```

**【4】ビルドの実行**

プログラムの追記ができたら,「ビルド」メニューのビルドを選択するか，アイコンをクリックすると，コンパイル／リンク処理がはじまる．エラーが発生すると，下部の出力欄に赤いマークが出るので，当該行をダブルクリックすると，エラーが生じた行へジャンプしてくれる．適宜修正する．

正常に終了すると,「Build Finished 0 Errors, ‥」と最終行に表示される．いくつか Warning(警告) が出るかもしれないがリンク処理で出たものは気にしなくてよい．

生成された機械語実行コードは,「X 月 Y 日 ¥RobotJikken¥Release」フォルダに RobotJikken.mot というファイルで作成されている（生成されたファイルは，モトローラ S-レコード形式と言われる．広く普及しているフォーマット形式で，中身は 16 進表示のテキスト形式である）.

**【5】作成された機械語コードをロボットにダウンロード**

ダウンロードの前に，図 6 を参考に再度接続 (センサとの接続，PC との接続) を確認すること．

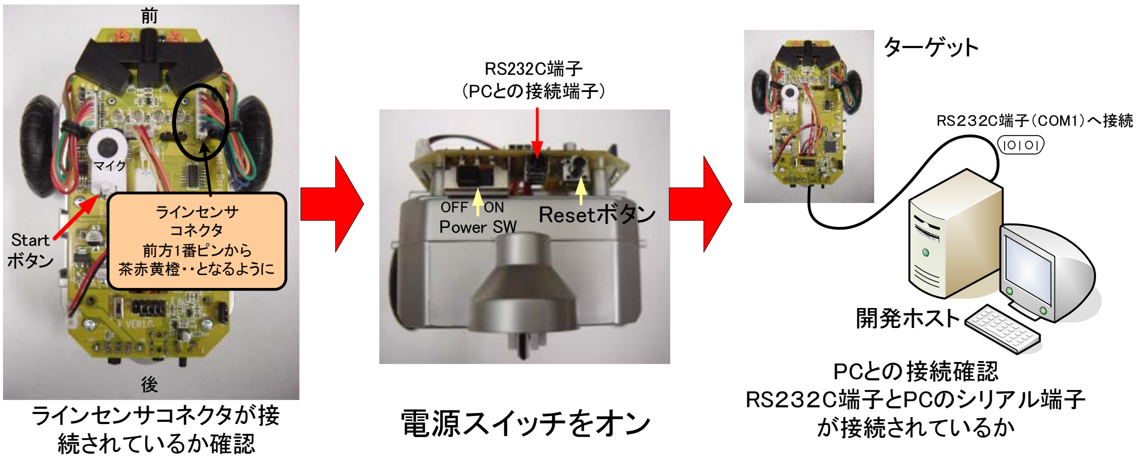



図 6: センサ接続の確認，電源オン，PC との接続確認

**Step1** **FDT の起動と設定**

デスクトップにある Flash 書き込みツール FDT ソフトウェア (Flash Development Toolkit) を起動して，設定する．起動時の unsupported version であるとのウィンドウは OK をクリックする．次の表示ウィンドウも「Cancel」をクリックする．

「Device」メニューの最下行「Configure Flash Project」を選択する．ウィザード形式で設定できる．最初にターゲット MPU を指定する Select Device に「H8/3672F」が選ばれているか確認 (そうでなければプルダウンメニューから選ぶ) して次へをクリック．次に，MPU（ロボット）の接続先設定を確認：今回は RS232C シリアル接続であるため，実験 PC では「COM1」が選ばれていれば良い．あとは，すべてデフォルトで「次へ」をクリック，最後に「完了」ボタンで設定終了．

**Step2** **書き込みプログラムを開く**

書き込む S-レコード形式の実行プログラムを開く.「File」メニューの「Open An S-Record...」を選択する．ビルドで作成された「X 月 Y 日 ¥RobotJikken¥Release」フォルダ内の RobotJikken.mot ファイルを選択して開く（以前開いたものが表示されている可能性があるので，**ファイルの場所が各班のフォルダであるか確認すること!!**）．すると，16 進表示の書き込むべきメモリイメージが表示される．

**Step3** **プログラムの書き込み**

ロボットの電源スイッチがオンであることを確認後，図 7 を参考に**ロボット上の BootMode スイッチを切換えて，ブートモードに変更する**．赤色 LED が点灯するはずである．ブートモードとは，MPU のフラッシュメモリを強制的に書き換える起動モードである．ブートモードスイッチを切換えたら，図 6 真中の図に示しているリセットボタンを押してリセットをかける．

いよいよ書き込みである．「Device」メニューの「Connect to Device」を選択する．「Connection complete」と下部の出力欄に緑字で表示されれば接続は OK である．「Error No 15024: Boot failed」などと赤字でエラーメッセージが表示された場合には，ブートモードに切換えているか（図 7），図 6 真中の図に示しているリセットボタンを押したか，あるいはきちんと PC とロボットが接続されているか（ロボット側の RS232C 端子が奥まで挿入されているか）を確認して再度「Connect to Device」を試す．

次に「Download Active File」を選択すると，今開いているファイルがロボットの MPU 上にダウンロードされる．すべて上手くいくと下部の出力欄に「Image successfully written to device」と表示される．その後，FDT 上で**「Device」メニューの「Disconnect」を選択**して，接続を解除しておくこと．後でまた FDT を使うので最小化しておく．

最後にブートモードから通常のモードへ戻して，リセットボタンを押す．

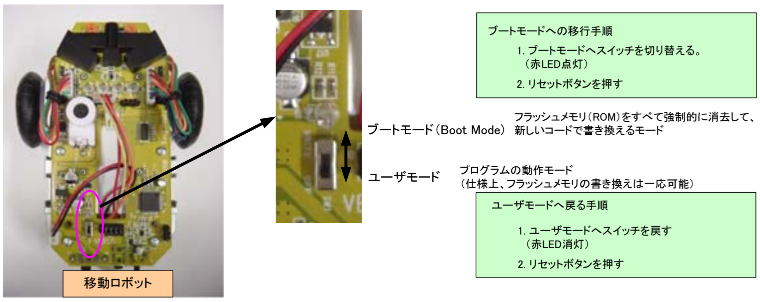



図 7: ブートモードの設定

## 【6】ロボット上の Start ボタンを押して動作させてみる

4 連 LED が全灯／消灯を 1 秒毎に繰り返していればダウンロードしたプログラムで動作している．もう一度 Start ボタンを押すとリセットされ（割り込み処理で行っている），再度 Start ボタン待ちの状態になる．電源を落としてもプログラムは保持されているため，電源再投入後も問題なく動作する．

## 5　RAM 領域に一部実行コードを置いてプログラムを動作させる (RAM 版)

プログラムの開発段階では，いちいちフラッシュメモリ上のプログラムを総入れ替えしていると手間で時間もかかり，面倒になってくる（今回の実験でもそうである）．また，フラッシュメモリに書き込み回数の制約があることも理由である（データシート上では書き込み保証回数 1000 回程度，実際にはもっと多くても OK のようだが,,）そのため開発段階では，上手に RAM 領域を利用することが良く行われる．ルネサスでは，組み込み装置向けの OS のような役割をなす**モニタプログラム**と呼ばれるプログラムを提供して実行プログラムを RAM 上へロード／実行したり支援するコードやデバッグに便利なシミュレータ等も提供している．

搭載している MPU は RAM 領域が 2KB と小さいため，簡易的な方法をとることにする．今回の実験では，プログラムの main 関数の中の「ここから ‥ 記述」〜「ここまで」の部分を開発していくことが主な作業となる．そのため，該当するプログラムコード部分のみ RAM 上に格納することを考える．つまり，その部分のコードの変更であれば，RAM 上のプログラムを置き換えるのみで良くなる（RAM は何度も自由に書き換え可能）．

**【1】main() 関数周辺の記述修正**

「ここから ‥ 記述」〜「ここまで」の部分を関数化するために，以下のようにプログラムを修正する．main 関数の後ろに追記する user_main 関数のように`#pragma section`で囲まれたプログラム領域が RAM 領域に置かれるコードを意味する．

```
.
.

void user_main(void);                                    ⇐ プロトタイプ宣言を追記


int main(void)
{
        .
        .
         while(1){
                .
                .
                ////////////////////////////////////////////ここから
                // ここからロボットの動作プログラムを記述
                ////////////////////////////////////////////

                user_main();                             ⇐ 関数呼び出しに修正

                ////////////////////////////////////////////ここまで
         }
}

// user_main() 関数のコードは RAM 領域に配置                  ↓ user_main 関数を追加
#pragma section USR
void user_main(void)
{
        //4 連 LED の全灯／消灯繰り返し：先ほどのコードをコピー
        while(1){
                Led_Pat( 0xF );// 4 連 LED をすべて ON
                Wait( 100 ); //100 x 10ms = 1 sec の待ち
                Led_Pat( 0x0 );// 4 連 LED をすべて OFF
                Wait( 100 );
        }
}
#pragma section
 .
 .
```

以上の修正の後，ビルドする．そして前章の項目【5】**Step3** の書き込み手順を実行する．その際，FDT をアクティブにすると「新しいファイルで読み込み直すか（ ··· Would you like to reload it?)」と聞かれるので OK とすれば，わざわざファイルを開き直す必要はない．プログラムを MPU のフラッシュメモリへダウンロードする際，**ブートモードに変更するのを忘れない**，また**書き換えを終えたら，戻すのも忘れない**.（うまくいかない場合，一旦「Disconnect」して再トライ）．ダウンロード後，FDT 上で**「Device」メニューの「Disconnect」を選択**して，接続を解除しておくこと．

書き込みを終えたら，ブートモードから通常の起動モードに戻して，リセットボタンを押す．そして，ロボット上の Start ボタンを押してみるとどうなるだろうか．この段階では，LED の全灯／消灯動作が上手くいかないはずである．ROM 上へはプログラムが転送されたが，上記で追加した `user_main()` 関数のコードは，まだ RAM 上に置かれていないからである．

**【2】user_main() 関数部分の機械語コードを RAM にダウンロード**

main.c プログラムには，動作中いつでもシリアル接続経由で RAM に置くべきプログラムを受け取れるようにしてある（`int_sci3()` 割込関数や `Down_Data()` 関数で行っている）．そのため，RAM にダウンロードする際には，わざわざブートモードに変更する必要はない．

ビルド時の最終の処理段階において，RAM に置くべき機械語コードのみ抜き出す処理を行っているため,「`X 月 Y 日 ¥RobotJikken¥Release`」フォルダ内に RobotJikken.mtr というファイルも存在するはずである．これは RobotJikken.mot 機械語コードの user_main() 関数部分のみを抜き出したものである．

**Step1**

RAM への書き込みツール（本実験のために独自に作成したもの）がデスクトップに SysJikken.exe として置いてあるので起動する．

**Step2**

起動後,「プログラム選択」ボタンを押して，RobotJikken.mtr ファイルを選択する（☞**その際，ファイルの場所が各班のフォルダであるか確認すること!!**）．

**Step3**

最後に「RAM へ転送開始」ボタンを押す．ロボットがコードを受信中，ロボット上の 4 連 LED が全部点灯するようにしてある.（プログラムが短い場合は一瞬全灯するのみである）

☞**補足**: シリアルポートの設定は COM1 をデフォルトにしているため，特に設定変更の必要はないが，もし上手く転送ができない場合，FDT 上で「Device」メニューの「Disconnect」を選択して，接続を解除してあるか確認すること．同時には接続できないため.

**【3】ロボット上の Start ボタンを押して動作させてみる**

4 連 LED が全灯／消灯を 1 秒毎に繰り返していれば，前項【2】でダウンロードしたプログラムが動作している．次に示すプログラムのように user_main() 関数の Wait 時間を 100 から 10 に変更して，再度ビルド処理を行い，前項【2】で起動している RAM への書き込みツールの「RAM へ転送開始」ボタンを押すと，新しい user_main() 関数のコードに即座に置き換えることができる．Start ボタンを押すとどうなるか．LED の点滅が速くなっていれば，コードが置き換えられたことを意味する．

```
…
}
// user_main() 関数のコードは RAM 領域に配置
#pragma section USR
void user_main(void)
{
        //4 連 LED の全灯／消灯繰り返し
        while(1){
                Led_Pat( 0xF );// 4 連 LED をすべて ON
                Wait( 10 ); //10 x 10ms = 0.1 sec の待ち        ←１０に変更
                Led_Pat( 0x0 );// 4 連 LED をすべて OFF
                Wait( 10 );                                    ←１０に変更
        }
}
#pragma section
```

これ以降は，コードの記述，ビルド,「RAM へ転送」の繰り返しで目的の動作となるようプログラムを開発していけば良い．ただし，RAM へ転送したコードの部分は電源を落とせば消滅する．

安定に機器を動作させる機械語コードの最終版が出来上がれば，ソースコード中の`#pragma section`の 2 行を削除（あるいはコメントアウト）してビルドして，前章の項目【5】の手順でフラッシュメモリの総入れ替えを行えば，すべてのプログラムコードを ROM 上に格納して動作させせることができる．

# 6　第 1 日目の実験課題

**ロボットにライントレース動作をさせてみよう！**

白地の床面に引かれた黒ラインをたどって（トレースして）動作させる．

1. **走行面（白地）に引かれた黒線を見分けるセンサの構成**

　ロボットの下部には，ラインを読み取るための３つの CdS セルと呼ばれる光導電効果を利用した半導体フォト・センサが備えられている．CdS セルは硫化カドミウムの光の吸収特性を利用した化合物半導体であり，明るいと抵抗値が減少，暗くなると抵抗値が増える特性を持っているため，明るさセンサとして身の回りの機器や外灯などのセンサとして広く利用されている．

　ロボットには，３つの CdS セルと合わせて赤色 LED が隣接して備えてあり，床面からの反射光が主に入射する構成となっている．そのため，白地面と黒地面で入射光量が変化するために各 CdS セルが黒地ライン上にあるのか，そうでないのか検出することができる．

2. **CdS セルと MPU との接続構成，及び AD 変換による量子化とその表示**

　CdS セルは 5V の定電圧源に 10KΩ の抵抗と直列に接続されている．図 8 中の A 点の電位が AD 変換で MPU に取り込まれるように接続している．MPU に内蔵されている AD 変換は，０～５ V の電圧を 10 ビットの 2 進コード（すなわち，0～1023 の整数値）のデジタル量に変換する．明るい場合（すなわち白地の床面での反射光）には CdS セルの抵抗値が低下して，A 点の電位は 0V に近くなる，すなわち変換された整数値も小さくなるはずである．逆に暗い場合（黒地ライン上での反射光）には抵抗値が上昇し，5V に近くなり，AD 変換して取り込んだ値は大きくなることが予想される．

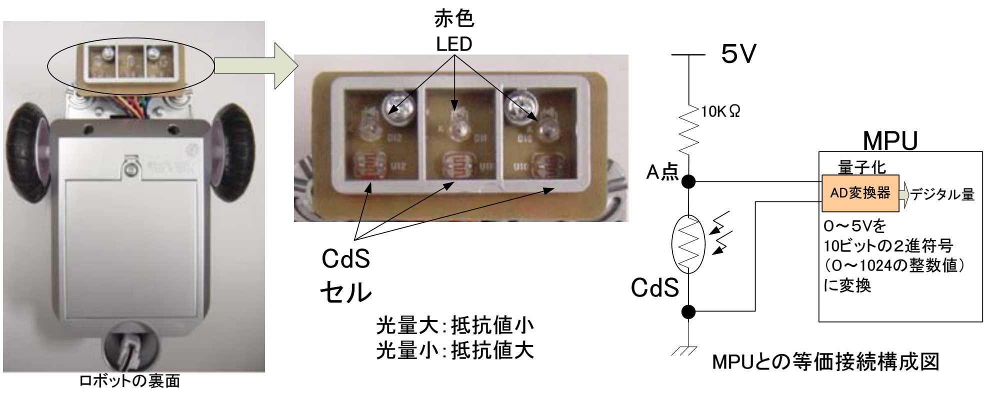



図 8: CdS セルの外観と接続構成図

　まずは，取り込まれた AD 変換後の整数値が，白地上，黒ライン上でどのような値を取るか調べてみる．そのための処理を記述したものを次頁の枠内に示しているので，`user_main()` 関数を書き換える．そして，ビルド後，SysJikken.exe の「RAM へ転送開始」ボタンを押して，新しいコードを RAM 上に転送する．ロボット上のスタートボタンを押してプログラムの動作を開始する（スタートボタンをもう一回押すとプログラムは再度スタート待ちの状態となって停止する）．

**— 確認プログラムの説明 —**

プログラム中の以下の関数は，センサ値を AD 変換して取り込むための関数である．

```
int get_line_sensor( int channel)
```

引数の channel の部分には，0,1,2( 0=右センサ，1=中央センサ, 2=左センサの値) のいずれかを指定すると，戻り値として，その AD 変換値が返される仕様である．またロボット上になんら表示装置がないため，その値を確認するためにその AD 変換値を RS-232C 通信を通して PC に送信する記述を加えている．

PC 画面に戻って，SysJikken.exe 上の「受信開始」ボタン押すことで，ロボットから送信されてきた文字列が右欄の「受信端末」上に表示されることを確認する．

右センサ，中央センサ, 左センサの順で変換値が表示されるはずである．ロボット位置を変えて，白地上，黒ライン上で値がどのように変化するか調べなさい．そして，白地面と黒地面であるかを検出するための，しきい値をここではとりあえず中間値（白地，黒地でのセンサ値の中央値）として求めなさい（しきい値：この値以下は白，そうでなければ黒ライン上と判断するための値）．端末への表示を終えるには,「受信開始」ボタンをもう一度クリックする．

**各センサ AD 変換値の結果**

| | 右センサ | 中央センサ | 左センサ |
|---|---|---|---|
| 白地面 | | | |
| 黒地面 | | | |
| しきい値 | | | |

**ラインセンサの読み取り値を確認するコード**

```
#pragma section USR
void user_main(void)
{
        int line_r, line_c, line_l;

        //ラインセンサの読み取り値を確認するコード
        while(1){
                Wait( 100 );//1s の待ち
                //各センサの AD 変換値を取得
                line_r=get_line_sensor(0);//右センサ
                line_c=get_line_sensor(1);//中央センサ
                line_l=get_line_sensor(2);//左センサ
                //各値を RS-232C 経由で PC に送信
                print_int( line_r );    //line_r を送信
                print_string( "\t" );  //タブを送信
                print_int( line_c );    //line_c を送信
                print_string( "\t" );  //タブを送信
                print_int( line_l );    //line_l を送信
                print_string( "\r\n" );//改行コードを送信
        }
}
#pragma section
```

3. **トレースしながら走行するために，読み取った値に応じてロボットをどのように動作させるか**

ロボットを前進，左右旋回させる関数はあらかじめ用意されている．章末の付録 B を参照すること．前進には，`Act_Fwd()` 関数，左右旋回動作には `Act_Rot_R()` 関数や `Act_Rot_L()` 関数，あるいは `Act_Turn_FR()` 関数や `Act_Turn_FL()` 関数を使うと良い.（引数の指定は付録 B の表を参照）

☞**動作アルゴリズムのヒント**

次のサンプルコード内に示すような `if` 文で比較的簡単に記述可能である．if 文の条件式の部分は，`line_r`，`line_c`，`line_l` の各センサ値とさきほど決めたしきい値との比較式が入る．

たとえば，次に示すようにセンサ値に応じて動作を if 文で切り替えていけばよい．**「中央センサが黒ライン上にあれば，前進する．右センサが黒ライン上にあれば，右旋回動作を行う．逆に左センサ上であれば左旋回を行う.」**これらの動作を短い時間間隔で繰り返すと良い．サンプルコードでは，Wait 関数

の引数を３に変更して，約 30ms 毎にセンサ値をチェックして，適宜動作を変更することになる．ロボット動作関数に前進する距離や回転角度を指定する必要があるが，動作は約 30ms 毎に更新されるため，距離／角度は無限に前進／回転となる値を指定すれば良い（付録 B 参照）．

前述の考え方だけでは，ラインから完全に脱線した際に復帰するための動作が明確化されていないため，脱線しない程度にモータの出力を適宜抑える必要がある．完全に脱線した状況まで配慮しながらアルゴリズムを考えると速いライントレースが可能である．余裕があれば上述したアルゴリズムにこだわらずに，独自のアイデアでトレースできるようにしても良い．

✎**注意) ロボットが机上から落下しないよう細心の注意を払うこと！**

**ライントレース動作コードのサンプル**

```
#pragma section USR
void user_main(void)
{
        int line_r, line_c, line_l;

        while(1){
                Wait( 3 );//30ms 毎に動作を修正するため，ここで待ち
                //各センサの AD 変換値を取得
                line_r=get_line_sensor(0);//右センサ
                line_c=get_line_sensor(1);//中央センサ
                line_l=get_line_sensor(2);//左センサ
                //
                //ここから動作を記述していく
                if(.... ){                     ←ラインセンサの値に応じて
                 ....                             直進や旋回等の動作
                }else if(.... ){                  を切り換える
                 ....
                }else if  ....
        }
}
#pragma section
```

✎ **補足**) ライントレース 1 周に要する時間を計測しながら，できるだけ速く周回できるよう改良してみなさい．上手くラインをトレースできない場合は，センサが白地上か黒地上にあるかを判別するためのしきい値設定に問題がある場合もあるので，その場合には再確認すること．また，しきい値の選定は白黒判別する際の感度とも考えることができる．黒ライン（約 1 cm 幅）がどの程度センサ上にかかっているかでセンサ出力値は変化するためである．

たとえば，白地上でセンサ出力値が 500, 黒地上で 700 だったとすれば（個体差があるため各ロボットで値は異なる），その中間の 600 をしきい値とすることは，単純に考えれば，黒ラインに半分ほど差し掛かった時点で黒と判別すると考えればよい．しきい値を変えることでライントレースの挙動は変ってくるので，適宜変更してみることも意味がある．

## 4. プログラムの印刷

✎**注意）main.c を全部印刷しないこと！量が多いため**

印刷は `user_main()` 関数の部分のみで良い．`user_main()` 関数の部分をマウスで選択した状態にして，ファイルメニューの印刷を実行する．印刷範囲が「選択した部分」となっていることを確認後，「OK」ボタンをクリックする．

# 7　第 2 日目の実験課題

**光センサ信号のフィードバックによる位置決め制御**

PSD（光スポットの位置検出素子）を利用した簡易光センサを接続し，障害物と一定距離を保つ制御を行う．そして，ロボットに障害回避の動作をさせてみる．

1. **PSD センサの接続**

   ロボットに PSD( Position Sensitive Detector) センサを搭載する．図 9 に示すように「ラインセンサコネクタ」を慎重に取り外し，PSD センサのコネクタに差し替える．赤線が 1 番ピン（ロボット前側）となるように，**コネクタの向きに注意して差し込むこと**．PSD センサはロボットの前方へ向けて，マジックテープで図中のように固定する．




図 9: PSD センサの接続と等価接続図

2. **PSD センサの出力特性を調べる**

   PSD センサは SHARP 製の GP2D12 であり，とても安価で簡易的な距離センサとして利用できる．原理は図 10 に示している．2 つのレンズが外観から確認できるが，片側から LED からの出射光，対象物からの拡散反射光がもう片方のレンズを通して PSD センサに集光される．対象までの距離に応じて，PSD 素子上に集光される位置が変化する．PSD 素子は，光スポットの位置検出素子であり，フォトダイオードの表面抵抗により，素子上の光スポット位置に比例した電圧を出力してくれる．その電圧はセンサ内部の増幅回路を経て，図 9 右に示すように MPU の AD 変換器に接続されている．

   図 10 に簡単な理論関係を示しているが，対象までの距離にほぼ反比例した電圧出力が得られる．センサのデータシートを章末の付録 C に掲載しているが，その Fig.6 のような出力特性となる．実際にどのような特性となるか，本実験ロボットで特性を調べてみる．

   Step.1

   第 1 日目のフォルダ（デスクトップの光システム実験 2 の AM or PM の X 月 Y 日）を丸ごとコピーして同じ場所に今日の日付「X 月 Z 日」として用意する．

   Step.2

   `user_main()` 関数を次に示すコードに置き換える．`get_ad()` 関数は，AD 変換された値を取得する関数であり，引数は接続先のチャンネル番号 (今回の接続では 2 となる) である．戻り値が変換値（10 ビット 2 進コード：0～1023 の値）である．それを電圧換算して，第 1 日目の実験同様に，その数値を PC に送信するための記述を加えている．

   ビルドした後，ロボット上のプログラムが他の班の実験によって書き換えられている可能性があるので，4 章の項目【5】の Step3 を参考にして FDT によるフラッシュメモリ（ROM）の総書き換えを行う．次に，第 1 日目の実験同様に SysJikken.exe による「RAM への転送」を行うことでプログラムが動作す

るようになる．なお，FDT によるフラッシュメモリ（ROM）の書き換えは，ここで一度行えば十分であり，これ以降は `user_main()` 関数の変更のみであるため，ビルドを行ってプログラムの変更を行ったとしても，SysJikken.exe による「RAM への転送」手順のみでプログラムの更新が行える．

ロボット上のスタートボタンを押して，動作確認を行う．SysJikken.exe 上の「受信開始」でセンサ電圧が「受信端末」に表示されることを確認しなさい．

**PSD センサの出力電圧を PC に送信して特性を調べる**

```
#pragma section USR
void user_main(void)
{
        int psd_data;
        float V_psd;

        while(1){
                Wait( 100 );//1s 毎に値を確認
                //PSD センサの AD 変換値を取得
                psd_data = get_ad(2);
                //AD 変換値を電圧に換算: 0～5V が，1024 段階（2 の 10 乗）に変換される
                V_psd = (float)psd_data / 1024.0 * 5.0;
                //PC に電圧を送信
                print_float( V_psd );
                print_string( "\r\n" );
        }
}
#pragma section
```

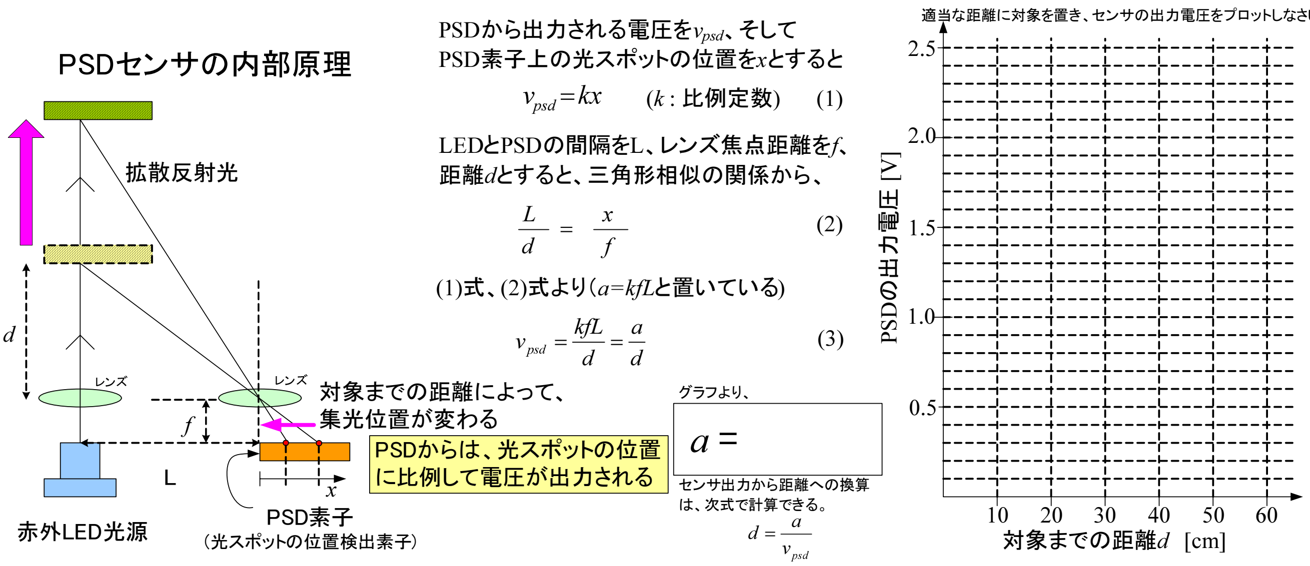



図 10: PSD センサの出力電圧と対象までの距離の関係

Step.3

適当な位置（センサの測距範囲：10cm～80cm）に適当な対象物を設置して，表示される電圧を確認する．何点か図 10 右のグラフにプロットしてみなさい.（付録 C のデータシート Fig.6 とも比較しなさい）．一つの測定距離に対して，何点（5 点程度）か電圧値サンプルをメモし（受信端末上で選択して，右クリック，コピーでも良い），エクセル等でその平均値を出してから（無ければ電卓），プロットしなさい．まれに明らかなノイズと思われる大きな値が出るが，そのような値は測定サンプルから除いて計算する．プロットを滑らかな曲線で結び，3 点ほどの位置で反比例定数 $a$ を算出して，その平均値を図 10 空欄に記入しなさい．

3. **比例制御（P 制御）による位置決め制御**

簡単な比例制御により，対象までの距離を一定に保つ位置決め制御を行ってみる．前項までの評価により，以下の式で対象までの距離が算出できる．

$$d = \frac{a}{V_{psd}} \tag{1}$$

目標とする対象までの距離を $d_{cmd}$ と置いて，現在の対象までの距離 $d$ と目標との誤差を適当な比例制御ゲイン $K$ を掛けてモータ駆動の制御入力 $u$ とする．つまり，以下を計算してモータ駆動の制御入力を決める．

$$u = K(d_{cmd} - d) \tag{2}$$

求められた結果を今回はそのままモータの出力値として与えて，両車輪を同じ入力で制御する．最終的に，モータ出力を決める Motor_L()/Motor_R() 関数（章末付録 B を参照）を使って，前進／後進動作で対象までの距離を保つフィードバック系が構成できる．(マイナス符号をつけているのは，対象との距離が増える方向とロボットの前進正方向が逆向きであるため)

```
Motor_L( -(int)u );
Motor_R( -(int)u );
```

センサ特性に用いた先ほどのプログラムを修正して (PC への送信部分は削除して良い：print_··· の関数部分)，以上に述べた位置決め制御動作を行えるようにしなさい．なお，制御周期（モータに指令を出す間隔＝ while ループの繰り返し周期）は，約 10ms 程度となるように Wait 関数の引数を 1 に変更する．また，制御ゲイン $K$ は，ほど良い応答特性となるよう適宜調整しながら実験的に定めなさい．この制御系では，摩擦などの補償を行っていないため，あまり小さいゲイン $K$ では，目標の距離に達する前に止まってしまったり，とても遅い追従動作となる．逆に $K$ を大きく設定すると，追従応答は速くなるが，大きすぎると，過大なオーバーシュート（行き過ぎ）が生じたり，また，モータの限界出力を越える指令 $u$ が生成されてしまうことでもオーバーシュートが増大する．

✎ 以下の $K$ の値における応答（前方の対象物を動かした際の応答）をメモし，$K$ の値を決めなさい

| $K$ の値 | 追従の速さを評価 | オーバーシュートがあるか | ほぼ目標距離に達しているか |
|---|---|---|---|
| 0.5 | | | |
| 1.0 | | | |
| 2.0 | | | |
| 4.0 | | | |
| 6.0 | | | |

✎ **注意) ロボットが机上から落下しないよう細心の注意を払うこと！**

4. **対象との距離が，30cm 以内の範囲にあるときのみ追従するようにする**

前方に何もなくても一気に前進してしまうので，ここでは対象物が一定の範囲に入ったら目的の距離（ここでは 20cm とする）に保つようにする．それ以外のときは静止する ( Motor_L()/Motor_R() 関数の引数をゼロ にする)．適当に if 文を利用して容易に変更可能である．

5. **前方の対象物を回避しながら動き回るように変更する**

前項では，前方に何もないと静止するようにしたが,「**状態１**：何もなければ前進する」ようにする．そして，30cm 以内に対象物が存在する場合,「**状態２**：一定の距離 20cm を保って，対象が居なくなるのを待つ」，状態２のまま 5 秒経過した場合,「**状態３**：90 度右（あるいは左）に回転動作を行う」の後,「状態１」に戻る．5 秒未満で対象が居なくなった場合は，即座に「状態１」に戻る．

☞**ヒント**

時間変数 `t` を導入すると良い．対象が 30cm 以内にあるときは，前項までの比例制御を行う，その際に時間を `t=t+0.01`（制御ループの周期が約 10ms なので）で更新して距離を保っている時間を管理する．もしそのまま時間が 5 秒経過すると，回転動作を行う (`Act_Rot_L()` または `Act_Rot_R()` 関数で良い．ただし，この場合，回転動作の完了を待つ必要があるので，その関数の直後に `Wait(0)` で動作の完了を待つ．引数がゼロの意味は付録 B を参照).

前方に対象が存在しない場合には，適当な出力で前進動作を続ける．その際は時間 `t` の更新をしないでゼロのままにしておく.（余裕がある班は，さらに拡張した動作をさせても良い.）

6. **プログラムの印刷**

✎**注意）main.c を全部印刷しないこと！量が多いため**

印刷は `user_main()` 関数の部分のみで良い．`user_main()` 関数の部分をマウスで選択した状態にして，ファイルメニューの印刷を実行する．印刷範囲が「選択した部分」となっていることを確認後,「OK」ボタンをクリックする．

# 8　レポートについて

レポートには作成したプログラム（第 1 日目，2 日目）を添付してもらうので，実験終了後，作成したプログラムを印刷するのを忘れないようにする．レポートには実験の目的，及び第 1 日目，2 日目の実験内容について，簡潔に書くこと．第 1 日目，2 日目で使ったセンサ（CdS セル，PSD センサ）について，その利用目的，簡単な原理，一般的にどのような用途で使われるセンサなのか調べて書くこと（テキストには敢えて詳しく書いていないので，各自調べること）.

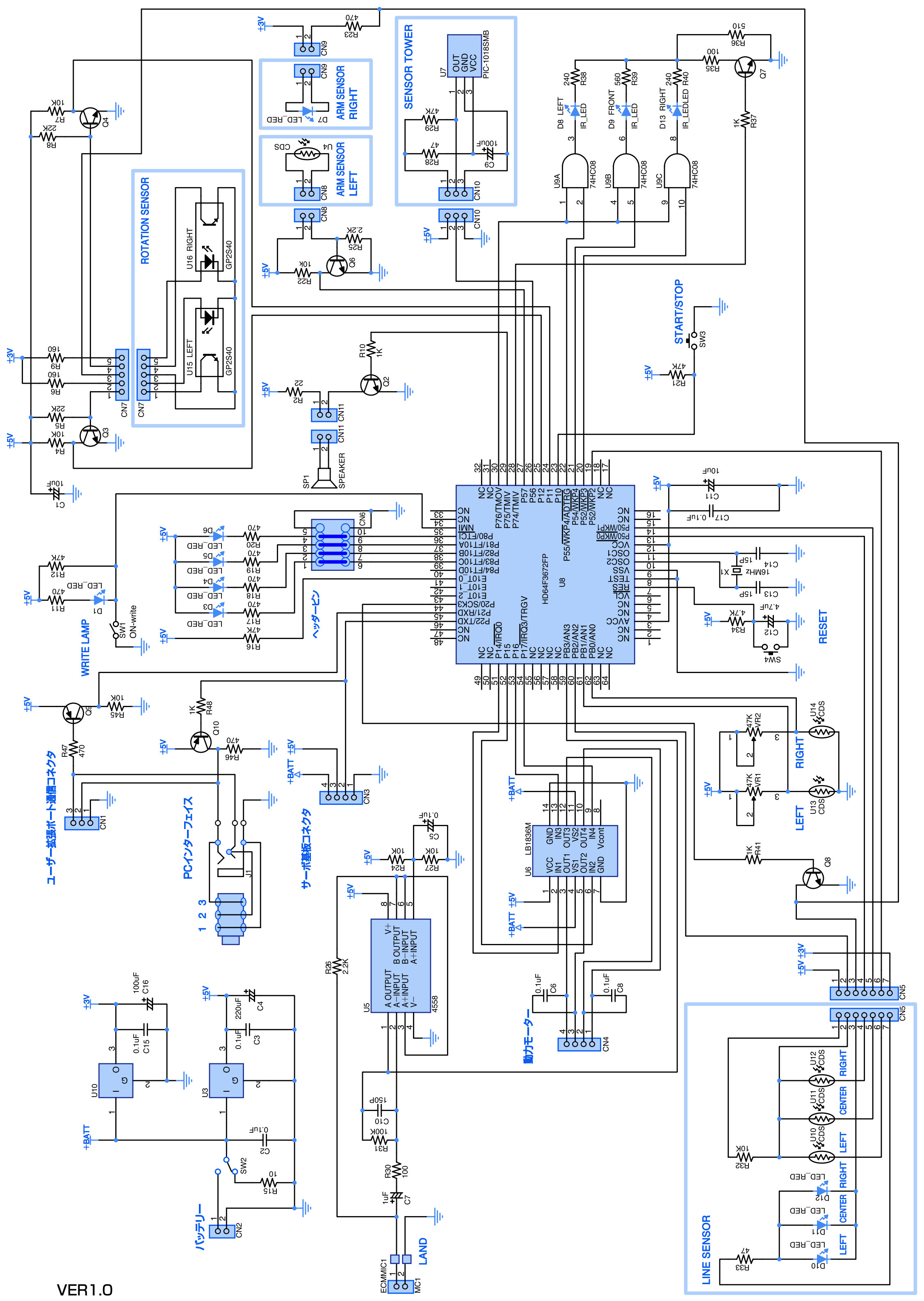
実験内容に関する考察では，第 1 日目：「今回自分達が試したトレースアルゴリズムを具体的に説明し，それを検討・評価しなさい，そして，より速く，正確にトレースするには更にどんな工夫が必要か考える」，第 2 日目：「比例制御ゲイン $K$ はどのように選んだか」及び「障害物を回避しながらロボットが行動するためには他にどんなセンサがあると良いか」についての考察も含めて書くこと．

最後にまとめを書くこと．実験目的，実験内容，結果，考察，まとめ等レポート全体を通して，読み手の立場に立ってまとめるよう十分に心掛けること．

# 付録

## ロボット内部の回路図

### 回路図




タンサーボーグ取扱説明書 (BANDAI) P.42 より抜粋

## ロボットの動作指令関数等の一覧

| | |
|---|---|
| **■ロボットの動作制御関係** | 補足）これらの関数は，動作の完了を待たずに戻る<br>動作完了を待つ必要がある場合は，Wait(0) を呼び出す（Wait 関数参照） |
| Act_Fwd(int p1,int p2) | 前進動作をさせる<br>p1:モータ出力 最小値：1 〜 最大値：31 の整数値を指定<br>p2:前進距離 [cm] 1〜200 の整数値を指定（ 250 以上で無限に前進） |
| Act_Bwd(int p1,int p2) | 後進動作をさせる<br>p1:モータ出力 最小値：1 〜 最大値：31 の整数値を指定<br>p2:後進距離 [cm] 1〜200 の整数値を指定（ 250 以上で無限に後進） |
| Act_Turn_FR(int p1,int p2) | 右旋回 前進動作をさせる<br>p1:モータ出力 最小値：1 〜 最大値：31 の整数値を指定<br>p2:旋回角度 [deg] 1〜360 の整数値を指定（ 400 以上で無限に旋回） |
| Act_Turn_FL(int p1,int p2) | 左旋回 前進動作をさせる<br>p1:モータ出力 最小値：1 〜 最大値：31 の整数値を指定<br>p2:旋回角度 [deg] 1〜360 の整数値を指定（ 400 以上で無限に旋回） |
| Act_Turn_BR(int p1,int p2) | 右旋回 後退動作をさせる<br>p1:モータ出力 最小値：1 〜 最大値：31 の整数値を指定<br>p2:旋回角度 [deg] 1〜360 の整数値を指定（ 400 以上で無限に旋回） |
| Act_Turn_BL(int p1,int p2) | 左旋回 後退動作をさせる<br>p1:モータ出力 最小値：1 〜 最大値：31 の整数値を指定<br>p2:旋回角度 [deg] 1〜360 の整数値を指定（ 400 以上で無限に旋回） |
| Act_Rot_R(int p1,int p2) | 右回転動作（その場で回転）をさせる<br>p1:モータ出力 最小値：1 〜 最大値：31 の整数値を指定<br>p2:旋回角度 [deg] 1〜360 の整数値を指定（ 400 以上で無限に旋回） |
| Act_Rot_L(int p1,int p2) | 左回転動作（その場で回転）をさせる<br>p1:モータ出力 最小値：1 〜 最大値：31 の整数値を指定<br>p2:旋回角度 [deg] 1〜360 の整数値を指定（ 400 以上で無限に旋回） |
| Act_Stop(void) | 停止 |
| **■モータへの直接指令関係** | |
| Motor_R(int p) | 右車輪のモータ出力を直接指定（負の値で逆方向の出力）<br>p:モータ出力 -31 〜 31 の整数値を指定 |
| Motor_L(int p) | 左車輪のモータ出力を直接指定（負の値で逆方向の出力）<br>p:モータ出力 -31 〜 31 の整数値を指定 |
| **■時間関係** | |
| Wait(int t) | 引数で指定する時間待ち<br>t: 10ms 単位で指定： 1=10ms，100=1s<br>t=0 のときのみ特別な待ちとなる： ロボット動作関係の処理の完了を待つ |
| wait_for_10ms_timer(void) | 無限ループの中に置くことで，繰り返し周期を 10ms にする<br>タイマー割込でカウントしているため，正確な制御周期にできる |
| wait_for_1ms_timer(void) | 無限ループの中に置くことで，繰り返し周期を 1ms にする<br>タイマー割込でカウントしているため，正確な制御周期にできる |
| **■ AD 変換関係** | |
| get_ad(int ch) | AD 変換器から変換値を取得， 戻り値： int 型の変換値<br>ch: チャンネル番号： 0〜3 の４つのチャンネルを指定可 |
| get_line_sensor(int ch) | ３つのラインセンサ（CdS）出力を取得（AD 変換器から）<br>戻り値： int 型の変換値<br>ch: 0〜2 の整数値を指定 0=右センサ，1=中央，2=左センサ |
| **■ PC への文字列送信関係** | |
| print_int( int val ) | 引数で指定される int 型データを文字列に変換して，PC へシリアル送信<br>val: 送信する int 値を指定 |
| print_float(float val ) | 引数で指定される float 型データを文字列に変換して，PC へシリアル送信<br>val: 送信する float 値を指定<br>（小数点以下第 2 位までを文字列に変換して送信） |
| print_string( char str[] ) | 引数で指定される文字列を PC へシリアル送信<br>str: 送信する文字列を指定<br>改行コードの送信の場合、"¥r¥n"と指定すると良い |

# GP2D12/GP2D15

## General Purpose Type Distance Measuring Sensors

## ■ Features

1. Less influence on the color of reflective objects, reflectivity
2. Line-up of distance output/distance judgement type
   Distance output type (analog voltage) : **GP2D12**
   Detecting distance : 10 to 80cm
   Distance judgement type : **GP2D15**
   Judgement distance : 24cm
   (Adjustable within the range of 10 to 80cm)
3. External control circuit is unnecessary
4. Low cost

## ■ Applications

1. TVs
2. Personal computers
3. Cars
4. Copiers

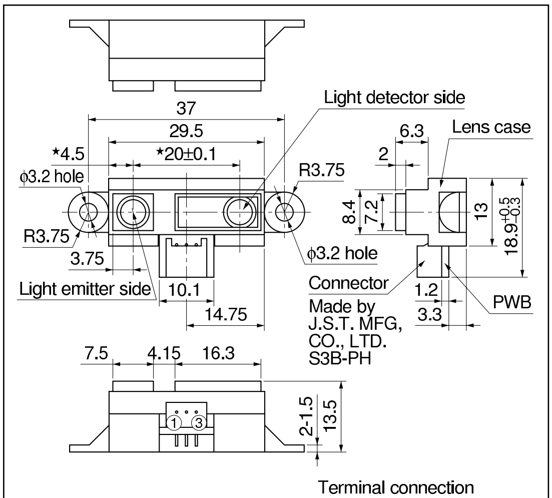
## ■ Absolute Maximum Ratings

(Ta=25°C, Vcc=5V)

| Parameter | Symbol | Rating | Unit |
|---|---|---|---|
| Supply voltage | $V_{CC}$ | –0.3 to +7 | V |
| Output terminal voltage | $V_O$ | –0.3 to Vcc +0.3 | V |
| Operating temperature | $T_{opr}$ | –10 to +60 | °C |
| Storage temperature | $T_{stg}$ | –40 to +70 | °C |

## ■ Outline Dimensions

(Unit : mm)




Notice In the absence of confirmation by device specification sheets, SHARP takes no responsibility for any defects that may occur in equipment using any SHARP devices shown in catalogs, data books, etc. Contact SHARP in order to obtain the latest device specification sheets before using any SHARP device.
Internet Internet address for Electronic Components Group http://www.sharp.co.jp/ecg/

## ■ Recommended Operating Conditions

| Parameter | Symbol | Rating | Unit |
|---|---|---|---|
| Operating supply voltage | $V_{CC}$ | 4.5 to +5.5 | V |

## ■ Electro-optical Characteristics

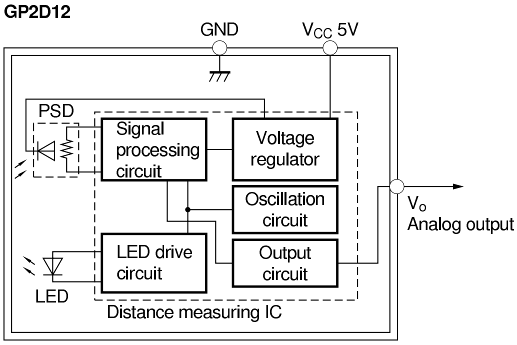
(Ta=25°C, $V_{CC}$=5V)

| Parameter | | Symbol | Conditions | MIN. | TYP. | MAX. | Unit |
|---|---|---|---|---|---|---|---|
| Distance measuring range | | ΔL | *1 *3 | 10 | – | 80 | cm |
| Output terminal voltage | GP2D12 | $V_O$ | L=80cm *1 | 0.25 | 0.4 | 0.55 | V |
| | GP2D15 | $V_{OH}$ | Output voltage at High *1 | Vcc –0.3 | – | – | V |
| | | $V_{OL}$ | Output voltage at Low *1 | – | – | 0.6 | V |
| Difference of output voltage | GP2D12 | $\Delta V_O$ | Output change at L=80cm to 10cm *1 | 1.75 | 2.0 | 2.25 | V |
| Distance characteristics of output | GP2D15 | $V_O$ | *1 *2 *4 | 21 | 24 | 27 | cm |
| Average Dissipation current | | $I_{CC}$ | L=80cm *1 | – | 33 | 50 | mA |

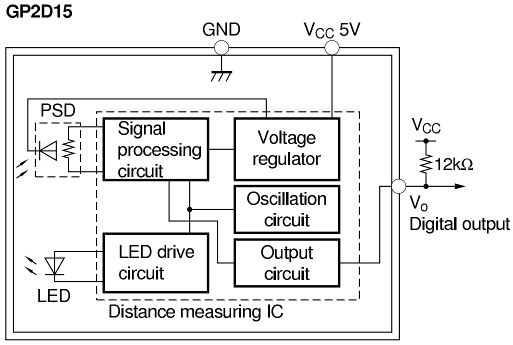
Note) L : Distance to reflective object.

*1 Using reflective object : White paper (Made by Kodak Co. Ltd. gray cards R-27 · white face, reflective ratio ; 90%).
*2 We ship the device after the following adjustment : Output switching distance L=24cm±3cm must be measured by the sensor.
*3 Distance measuring range of the optical sensor system.
*4 Output switching has a hysteresis width. The distance specified by Vo should be the one with which the output L switches to the output H.

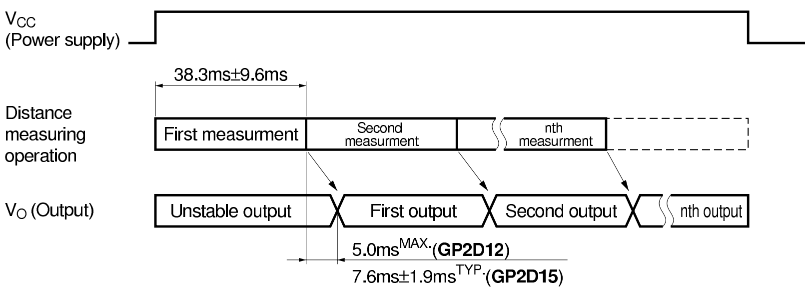
### Fig.1 Internal Block Diagram



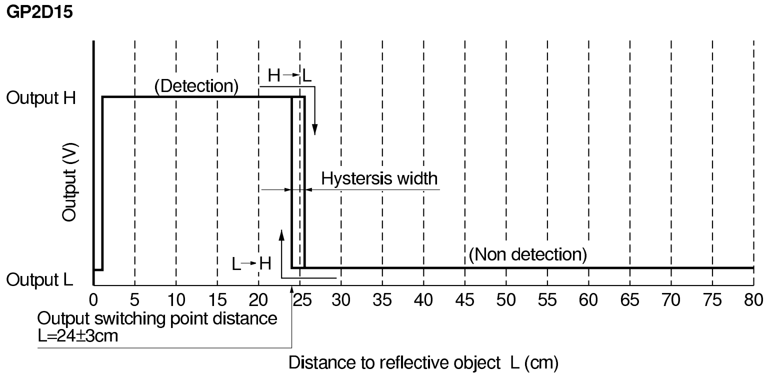

### Fig.2 Internal Block Diagram



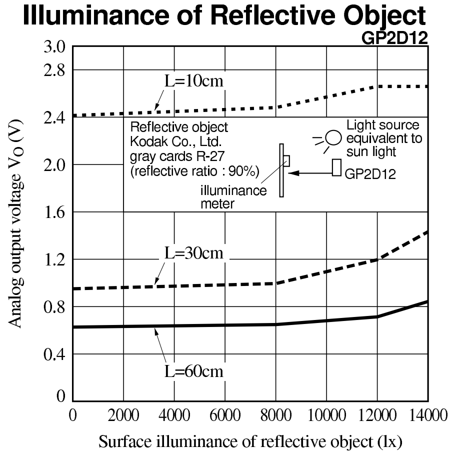

### Fig.3 Timing Chart



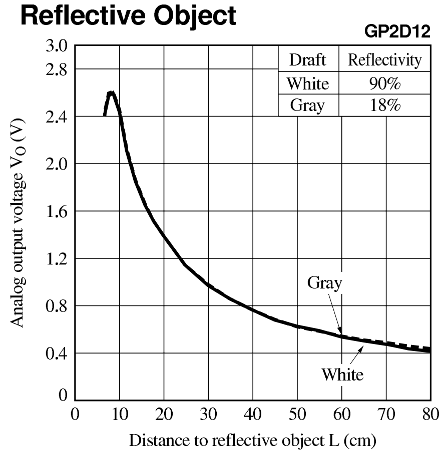
## Fig.4 Distance Characteristics



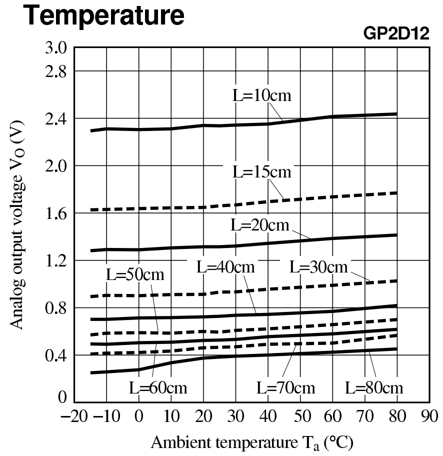

## Fig.5 Analog Output Voltage vs. Surface Illuminance of Reflective Object



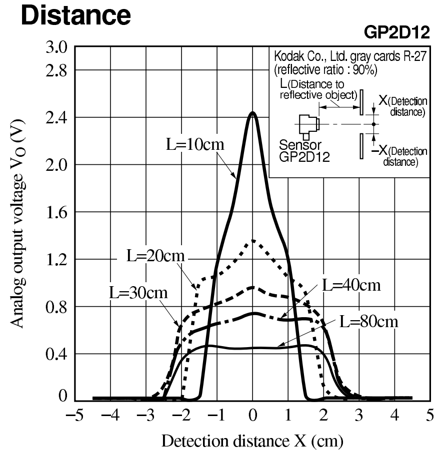

## Fig.6 Analog Output Voltage vs.Distance to Reflective Object




## Fig.7 Analog Output Voltage vs.Ambient Temperature




## Fig.8 Analog Output Voltage vs.Detection Distance



## C 言語リファレンス

| **while** | 繰り返し |
|---|---|

**書式** while( 式 1 )

**意味** 式 1 がループを継続する条件式である．式 1 を満たしている (真である) 限りループが継続する

**利用例** 例 1

```
i=0;
while( i< 10 ){
        （処理１）
        i++;
}
```

ループ前に i=0 が実行され，一回目の処理 1 の実行の前に条件式 i ＜ 10 が評価される．満たしていなければ，処理 1 は一度も実行されずにループは終了する．例 1 では満たしているので 1 回目の処理 1 は実行される．処理 1 の後に明示的に i++を実行しており，2 回目の処理の前に再度 i ＜ 10 が評価される．for 文との大きな違いは条件式の評価が繰り返し処理前に行なわれる点である．例 1 では結果的に処理 1 が 10 回実行される．

例 2

```
while(1){
        （処理２）
}
```

評価式である式 1 の値が常に 1 である．つまり条件式が偽となる 0 には決してなることはないため，無限にループを繰り返す．for(;;) の場合と同様にループの中断処理を行なう break 文と組み合わせて使われることが多い (break の項目参照). 組み込み機器などの制御機器では無限に動作を継続する必要があるため，while(1) が良く使われる．

| for | 繰り返し（ループ） |
|---|---|

**書式** for( 式 1 ; 式 2 ; 式 3 )

**意味** ループの開始前に式 1 が実行される．式 2 はループを継続する条件式．式 3 は繰り返しごとに実行される式.

**利用例** 例 1

```
for(i=0; i< 10; i++){
         (処理１)
}
```

ループ前に i=0 が実行され，処理 1 が一回実行されるごとに，i++が実行される（i++は i=i+1 と同じ）．その後，条件式である i ＜ 10 が評価され，満足していれば 2 回目の処理 1 が実行される．i ＜ 10 を満たしていなければループ終了．したがって，例 1 では処理 1 が 10 回実行されてループが終了することになる．

例 2

```
for(;;){
         (処理２)
}
```

これは特殊な例であるが，式 1，式 2，式 3 を全て省略すると，無限にループを継続する．ループの中断処理を行なう break 文と組み合わせて使われることが多い (break の項目参照).

| do ～ while | 繰り返し |
|---|---|

**書式** do { 処理 1 }while( 式 1 )

**意味** 式 1 がループを継続する条件式である．処理 1 が実行された後に式 1 でループを継続するか評価する．満たしている (真である) 限りループが継続する

**利用例** 例 1

```
i=0;
do{
        (処理１)
        i++;
}while( i < 100 );
```

処理 1 がまず実行される．その後，継続して繰り返すかを条件式 i ＜ 100 で判断する．満たしていなければ，do{ } 内が繰り返し処理される．満たしていない場合はループを終了する．この例 1 では最終的に処理 1 が 99 回実行される．

| break | 繰り返し（ループ）を中断する |
|---|---|

**書式** break;

**意味** do, for, while, switch 文を強制的に終了する

**利用例** 例 1

```
while(1){
   (処理１)
   if( value == 0) break;
}
```

変数 value の値が 0 であれば (if 文の使い方は次項参照), ループの条件に関わらず強制的に終了する．

| continue | 繰り返しの先頭に戻る |
|---|---|

**書式** continue;

**意味** do, for, while の繰り返し開始位置にジャンプする．

**利用例** 例 1

```
while(1){
   処理１；

   if( value == 0) continue;

   処理２;
}
```

変数 value の値が 0 であれば繰り返し処理の残りの処理 2 を実行せず，次の繰り返し処理に強制的に移る．

| if | 条件式の評価に応じて処理コードを変える |
|---|---|

**書式**　if( 式 1 ){　(処理 1)　}else{　(処理 2)　}

**意味**　式 1 が評価式を満たしている（真である）場合は処理 1 が実行され，偽であれば処理 2 が実行される

**利用例**

例 1

```
if(i==1){
    (処理 1)
}else{
    (処理 2)
}
```

変数 i の値が 1 であれば処理 1 が実行される (==は演算子の項目参照)．そうでなければ処理 2 が実行される．

例 2

```
if(i==1){
    (処理 1)
}
```

else 以降は省略することもできる．

例 3

```
if( i>=0  && i<=5 ){
    (処理 1)
}
```

&&を用いて，i ＞=0 かつ i ＜=5 というように AND の評価を行なうこともできる．

例 4

```
if( i!=0  || i!=5 ){
    (処理 1)
}
```

|| を用いて，i!=0 または i!=5 というように OR の評価を行なうこともできる．!=は「等しくない」の意．

例 5

```
if( i<5 ){
    (処理 1)
}else if( i<10 ){
    (処理 2)
}else{
    (処理 3)
}
```

else if により複数の条件を段階的に評価できる．i ＜ 5 を満たしていれば処理 1 が，満たしていなければ else if により i ＜ 10 を評価し，それを満たしていれば処理 2 を実行する．どれも満たしていない場合に処理 3 が実行される．

| **switch, case** | 条件式の評価に応じて処理コードを変える |
|---|---|

**書式**

```
switch( 式１ ){
  case 定数１:
    処理１
    (break;)
  case 定数２:
    処理２
    (break;)
  default:
    処理３
}
```

**意味** 式 1 と一致する定数があれば，その処理が実行される．一致するものがなければ default の部分の処理が実行される．ある case 文の処理を行なった後，それ以降の case 文の処理判定が必要なければ break で switch 文から抜けることができる．尚，式 1 は整数値になる式，定数の部分も整数値である必要がある．同様の処理は if 文を用いても記述することができるが，処理をある値の定数で分岐させたい場合には，switch 文の方が分かりやすく記述できる．

**利用例** 例 1

```
switch( value ){
   case 0:
     処理１;
     break;
   case 1:
     処理２;
     break;
   case 2:
     処理３
     break;
   default:
     処理４;
}
```

変数 value の値が 0 であれば処理 1，1 であれば処理 2，2 であれば処理 3，その他は処理 4 が実行されて switch 文を終了する．

| **goto** | ジャンプする |
| --- | --- |

**書式**　goto ラベル;

**意味**　ラベルを定義した個所にジャンプする．goto 文はとても便利な分岐方法であるが，むやみに使うとプログラムの流れを追跡しにくくなる．ほかに方法がない場合や特別な理由がない限り使わないようにすること．

**利用例**　例 1

```
my_func(){

   if(i == 1) goto label1;

   i=i+3;

label1:

   i=j+2;
}
```

例のようにラベルの定義にはコロンを使う．最初の if 文が真であれば label1:を定義した部分にジャンプする．switch や continue, 関数から戻る return，関数の定義，{} を活用すれば goto を使う必要はほとんどない．なるべく goto 文は使わない．

**演算子**

| | |
|---|---|
| == | 値が等しいかの評価を行なう. |
| ！= | 不等評価を行なう. |
| < | 大小関係の判定（小なり） |
| > | 大小関係の判定（大なり） |
| <= | 大小関係の判定（以下） |
| >= | 大小関係の判定（以上） |

**書式** 式 1 == 式 2， 式 1 <= 式 2

**意味** 2 つの式あるいは値を評価して，真であれば True, 偽であれば False を返す．if 文や while 文等の条件式の中で使われる

**利用例** 例 1

```
if( i==1 ) i=i+1;
if( j<=1 ) j=j+1;
if( k>=1 ) k=k+1;
if( m!=1 ) m=m+1;
if( n <1 ) n=n+2;
```

注意 間違えやすい例

```
if( i=1 ) break;
```

==とするところを=と間違えて記述してもコンパイル時にエラーにならない． この場合には i=1 の代入式として処理される．i=1 の値は真である（０でない）ため break 文が実行される．

| | |
|---|---|
| ++ | インクリメント演算子 |
| - - | インクリメント演算子 |

**書式** 変数++, 変数- -, ++変数， - -変数

**意味** 変数に 1 加える（＋＋の場合），1 減じる（- -の場合）

**利用例** 例 1

```
i=0; j=0;
i++;
j--;
val=++i;
val=i++;
```

++(- -) を変数の前に置くか，後に置くかで処理される順序が異なる．val=++i; の場合には，変数 val に i の値を代入する前に i=i+1 が実行される．val=i++; の場合には i の値を val に代入してから i=i+1 が行なわれる．

**論理演算子**

| | |
|---|---|
| **&&** | 論理 AND 演算子 |
| \|\| | 論理 OR 演算子 |

**書式** 式 1 && 式 2 , 式 1 || 式 2

**意味** &&: 式 1 と式 2 ともに真のときに真となる. || : 式 1 と式 2 のどちらかが真であれば真となる.

**利用例** 例 1

```
if( a==1 && b==1 && c==1) i=10;
if( x==1 || y==1 || z==1 ) i=0;
```

複数の条件式の値を評価することができる.

**ビット演算子**

| | |
|---|---|
| **&** | ビットごとの AND 演算子 |
| \| | ビットごとの OR 演算子 |
| ˆ | ビットごとの排他的 OR 演算子 |
| ˜ | ビットごとの反転 |
| **<<** | 左シフト演算子 |
| **>>** | 右シフト演算子 |

**書式** 例： 式 1 & 式 2 , 式 1 | 式 2

**意味** 式の値をビットごとに演算する

**利用例** 例 1

```
a1=0x00ff;
a2=0xff00;
a3= a1 & a2; // a3=0x0000 となる
a4= a1 | a2; // a4=0xffff となる
```

2 つの変数 a1 と a2 をビットごとに AND 演算，OR 演算している例.

例 2

```
a1=0x01;
a2=0x10;
a3= a1<<1; // a3=0x02 となる
a4= a2>>1; // a4=0x08 となる
```

2 つの変数 a1 と a2 をビットシフトしている例である.

**関数（以降の関数は本実験では使う必要はない）**

| | |
|---|---|
| **rand()** | 0〜32767 の間の擬似乱数を生成する |
| **srand()** | 乱数ジェネレータを初期化する |

**書式** rand(), srand( seed_value )

**意味** 擬似乱数の生成，seed_value は乱数の生成に使われる初期値

**利用例** 例 1

```
value=rand();
```

0〜32767 の間の乱数が生成され value に代入される．尚，いきなり rand() によって乱数を生成すると，毎回毎回同じ乱数値が生成される．そのため rand() を使う前に srand() を一度だけ呼び出して乱数系列のシード値を設定する必要がある．

例 2

```
srand( time(NULL) ); //srand はプログラムの最初で一回だけ呼び出せば良い．

value=rand();
```

この例はシード値を現在の時間から決めることで，毎回同じ乱数が生成されないようにした簡単な例である．

| **printf()** | 文字列を出力する |
|---|---|

**書式** printf(”.....”,....)
**意味** printf: 文字列を出力する.
**利用例** 関数の機能は豊富であるため，ここでは簡単な利用例のみ書いておく．詳しくは専門書を参考にして下さい

例 1

```
printf("My Name is CIST.\n");
```

”.....”で囲んだ文字列を出力する．最後の”\ n”は「改行しなさい」という特別な意味を持つ.

例 2

```
i=100;
printf("Answer= %d \n", i);
```

出力する文字列の部分に`%d`が挿入されている.「整数値 i の値を`%d`の部分に代入して表示しなさい」という意味となる.

例 3

```
value=0.1234;
printf("Answer= %f \n", value);
```

出力する文字列の部分に`%f`が挿入されている.「小数点数 value の値を`%f`の部分に代入して表示しなさい」という意味となる.

例 4

```
i=10;
value=0.1234;
printf("Answer= %d , Value = %f \n", i, value);
```

この例のように複数の代入をすることが可能である．この場合には「`Answer = 10 , Value = 0.123400`」のようにそれぞれ順に代入されて出力される.